MILLERS GRAPHICS

SEPTEMBER 1984

$1.50

THE SMART PROGRAMMER

There have been quite a few new developements in Software and Hardware for us 4A owners since the last newsletter. First off I would like to apologize for what I was led to believe was a reputable company. SCI TECH turned out to be a pretty lousy company when it came to customer service. They never returned any of the calls on the RAM DISK mentioned in the last newsletter and the engineer moved out of state. Just as well. There is, however, another RAM DISK on the market which is made by MYARC. It is a 128K-512K RAM DISK with built in Print Spooler. It has most of the other features we discussed in the past few issues. They are also about to release an New and Faster version of Extended Basic that will use the extra memory in their RAM DISK card. You can contact them at:

Myarc Inc.
P.O. Box 140
Basking Ridge. NJ 07920
(201) 766-1700

They have also released a disk based Disk Manager for their Double Density Disk Controller Card.

Morning Star has released a 128K memory card also that has many bank switching options to allow you to place RAM in the Cartridge and DSR spaces as well as many other configurations. They can be reached at:

Morning Star Software
4425 SW 109th Ave.
Beaverton. OR 97005
(800) 824-2412

Many people and companies are working on other items of hardware and software. There sure is a lot of support for the 4A right now. I think TI jumped the gun when they bailed out. its still the best home computer around.

We've finished that Extended Basic book and Cassette combination that we talked about quite some time ago (The Smart Way to Award Winning Programming). It is now entitled 'NIGHT MISSION'. The cassette contains a new Extended Basic helicopter rescue game and the book fully documents the program flow. The book also contains a chapter on the use of AND to help speed up your programs. Another chapter contains some new CALL LOADs. Night Mission is in our NEW catalog that is currently being mailed out to everyone on our mailing list.

I would like to thank everyone for the nice comments and reviews on Advanced Diagnostics and EXPLORER. Some of you have had a few questions on these so we will tackle them in the Q & A section.

There is a New 32K 16 BIT RAM console modification for our 4As. This mod will allow all of your Assembly programs to run as if they are in Scratch Pad RAM - very fast! By making all of the 32K Memory Expansion 16 BIT it really opens up the 9900 micro and allows it to run without WAIT states in between each and every byte! We've been told that the speed increase is any where from 20 to 50% FASTER. I've sent one our consoles in for this mod so I should have more on this in the next newsletter. To have one of your consoles modified just send it to:

TRS Systems
17885 Mount Elliott
Detroit, MI 48212
(313) 366-9088

Along with a check or money order in U.S. funds for $125.00. This upgrade has a 1 year warranty. Dealer inquiries invited.

GRAM KRACKER™

Boy, do we have something NEW for you! We have completed the prototype and testing of some new hardware that Millers Graphics is going to produce and it should be ready to ship by November 15. This unit plugs into the module port and it contains 48K of RAM and GRAM. That's right GRAM, the RAM complement to GROM. We call this unit the GRAM KRACKER^{tm} and we are VERY pleased with the versatility and power of this little unit. You will be able to plug ANY module into the GRAM KRACKER^{tm} and SAVE its contents to Disk or Cassette. Once saved, they can be loaded into the battery backed up GRAM KRACKER^{tm} and run.

It is very simple to operate. On the front panel there are 4 switches and a Reset button. The switches control the loader, Ram banks and write protect, and Enable or Disable of GRAMs 0, 1 & 2. The GRAM KRACKER^{tm} also contains sockets on its circuit board to allow you to expand its total programmable memory to 80K of RAM and GRAM. This gives you 16K of bank switched Cartridge ROM/RAM and 64K of Console and Cartridge GROM/GRAM. This means you can now modify the contents of GRAM 0, the system monitor, GRAMs 1&2, TI Basic, and GRAMs 3-7, in the cartridge. You can also modify the contents of Cartridge RAM >6000->7FFF (2 banks) to suit your needs. So now you can change the Title Screen, change menus, have true ascenders and descenders for your character sets. You can change the default screen and character colors for any module. You can change the default printer configuration for your modules. You can add many new CALLs to TI Basic or TI Extended Basic. You can override TI Basic and put something else in its place. This would allow you to have a menu with a couple of different modules to select from.

With a full 80K GRAM KRACKER^{tm} you can have a menu that includes Editor/Assembler, TI Writer, Extended Basic and 1 more single Grom chip cartridge like Adventure. Once you have the GRAM KRACKER^{tm} configured the way you want, and the modifications made to the software in it, you can save it all out to Disk or Cassette. Since the GRAM KRACKER^{tm} is battery backed up, you DO NOT have to reload it each time you turn on your computer!

Now you can take advantage of the powerful GPL Language that is built into our 4As. Through Assembly Language GPL Link you can execute GPL routines that YOU have set up. There are many routines in the interpreter that perform common tasks quite nicely. For example; the GPL code 07 20 00 will clear a 32 column screen and return to your Assembly program. This is the opcode for ALL >20 RETURN. Once you've loaded this opcode into GRAM just execute a BLWP @GPLLNK DATA >xxxx, where xxxx = the GRAM address where the opcode is. Since GRAM is auto-incrementing memory,like GROM, it has an added bonus in that it loves to store and retrieve DATA. So with only the Editor/Assembler module loaded into the GRAM KRACKER^{tm} you still have 32K of GRAM DATA storage AND 16K of bank switched cartridge RAM plus ALL of memory expansion to use. So that's 80K of Program and DATA area and we haven't even touched VDP Ram yet! With an 80K GRAM KRACKER^{tm} and Memory Expansion we now have 64K of GRAM, 16K of Cartridge RAM, 32K of Memory Expansion RAM and 16K of VDP RAM for a total of 128K of RAM & GRAM. Oh those clever folks at TI, if they would have only opened this up to us sooner! Who says the TI 99/4A isn't a programmers machine!

The GRAM KRACKER^{tm} has an additional 8K of programming built in to it for the SAVE, LOAD and EDIT routines, which are enabled and brought up on the menu with one of the switches. It also comes with a floppy diskette that contains some modifications and enhancements for Extended Basic. The GRAM KRACKER^{tm} will be produced in limited quantities at first and the quality will be guaranteed by us. We are not going to ramp up production into high gear only to have to file chapter 11 because of miss management or over zealous production. We want to build a Reliable, Long Lasting, Quality product. After this project is on the road we have many more items planned. And you thought the 99/4A was an orphan? Little Orphan Annie is more like it!

By the time you read this the GRAM KRACKER^{tm} will be ready to start production and as soon as it is ready to ship we will mail out brochures. The price will be in the 150 to 200 range, we're finalizing parts prices with the suppliers now. Finally the secret world of GPL is open to us to play, modify and program in, and is it a lot of FUN!

Q & A

We have received a number of questions on Assembly Language so lets look at a few of them. The first one concerns the use of a BL or BLWP that is followed by one or more DATA statements, ie:

```
BLWP @GPLLNK
DATA >0020
```

A BLWP or Branch and Load Workspace Pointer is known as a context switch. When this instruction is executed it grabs its New Workspace Pointer and PC (Program Counter) from a vector table or location. When the instruction BLWP @GPLLNK is assembled it is converted into the opcode >0420 >2100 for the E/A module. Then when the instruction is executed the micro will grab the new Workspace Pointer from >2100 and the new PC from >2102 and it will continue execution starting at the new PC. Also, when a context switch is performed the micro automatically places the old Workspace Pointer, old PC and old status into R13, R14 & R15 of the New Workspace. Then when a RTWP is executed the WS, PC & ST are restored and the program continues where it left off.

If a DATA statement follows the BLWP you will find a statement somewhere in the code that was branched to similar to;

```
MOV *R14+,R0
```

This will MOVe a word from the location pointed to by R14 (the old PC value which is now pointing at the DATA statement) into R0. Since this is an auto-incrementing word MOVe R14 will increment by 2 and point to the word after the DATA statement. Then when a RTWP is executed, since R14 was changed, the program will return to the word following the DATA statement. If the statement was a BL or Branch and Link ie:

```
BL    @XYZ
DATA  >1234
```

Then you will find or need an auto-increment MOVe from R11 since R11 holds the Return address (old PC) for a BL ie:

```
MOV *R11+,R0
```

will get the DATA following the BL statement and put it into R0, it will also increment R11 by 2 to point to the word following the DATA statement for the return. Both the BLWP and BL with DATA following them are just one method of transferring DATA or values to the BRANCHed to routine. However, on a BL statement you can just keep the DATA in one of the existing workspace registers since a context switch is NOT performed and the workspace is the same. For every DATA statement following a BLWP or BL you will need or find an auto-incrementing MOVe from *R14+ or *R11+.

Why doesn't Extended Basic's XMLLNK work properly?

The XMLLNK that is loaded into Low Memory Expansion by Extended Basic's CALL INIT is different than the XMLLNK loaded by the Editor/Assembler or the one in the Mini Memory. In the back of the Editor/Assembler manual on pages 415-418 you will find a list of XB Equates. Use the Equates from FADD through GVWITE for the DATA statement following BLWP @XMLLNK and XB's XMLLNK will work properly. I Don't know why TI changed so many of the routines in XB as compared to E/A. By the way, the Equates listed from FADD to NEXT are ABSOLUTE ROM addresses for ALL consoles! The equates that are a byte in length like CIF EQU >26 are XML Table pointers that use routines built into XB's Cartridge ROM.

How do you execute CALL FILES(1) in Assembly Language?

In the Disk Controller card there is a built in CALL FILES subroutine that can be CALLed from the Basics or executed from an Assembly Language program. To get to it you should use the DSRLNK routine followed by DATA >A (or DATA 10). Prior to executing the

```
BLWP @DSRLNK
DATA >A
```

you need to place a byte for the number of open files into location >834C. So, for CALL FILES(1) you place a 1 there, for CALL FILES(6) you place a 6 there. The valid range for the Assembly CALL FILES subroutine is 1 to 16. You will also need to set up a PAB in VDP Ram with a name length of 1 and the subroutine number of >16 ie:

```
PABDAT BYTE 0,0,0,0,0,0,0,0,0,1,>16
```

and place the address of the PAB's name length byte into >8356. So if your PAB is at VDP RAM >1000 the value to place in >8356 is >1000+9 or >1009.

If the routine was successful, there was enough free VDP Ram, then the byte at >8350 will be zero otherwise it will be >FF or

some other non-zero value. If you place a value that is out of the valid range (1-16) into >834C the subroutine will just abort and no action will be taken. One last thing, when a BLWP @ DSRLNK with DATA >A following it is executed the normal error reporting is not active. By this we mean that the error code is NOT placed in your R0 and the EQual bit is NOT set in the Status register on return unless there is a Link Error. A Link error can occur if the Name and Length bytes are not set up right in the PAB or if the pointer at >8356 is not pointing to the Name Length byte of the PAB in VDP. Here is a little example Assembly listing for the Editor Assembler module.

```
*--------------------------------------*
* CALL FILES(1) example for E/A Module *
* for XB use COPY DSKx.DSRLNK (page 13)*
* delete REF and add VMBW EQU >2024    *
*--------------------------------------*
       DEF  START
       REF  DSRLNK,VMBW

PABADD EQU  >1000
PABDAT BYTE 0,0,0,0,0,0,0,0,0,1,>16
PABLEN EQU  >B
GPLWS  EQU  >83E0
LENPTR EQU  >8356
DSRERR EQU  >8350
FILES# EQU  >834C
NEXT   EQU  >6A

WS     BSS  >20

START  LWPI WS
       LI   R0,PABADD      Set up PAB in
       LI   R1,PABDAT          VDP
       LI   R2,PABLEN          RAM
       BLWP @VMBW

       LI   R0,PABADD+9    Set up Len pntr
       MOV  R0,@LENPTR     in scratch pad
       LI   R0,>0100       Set up # files
       MOVB R0,@FILES#     in scratch pad
       BLWP @DSRLNK        Execute FILES
       DATA >A             (subprogram)
       JEQ  ERROR          Link error

       MOVB @DSRERR,@DSRERR  enough room
       JNE  ERROR            in VDP Ram?
       LWPI GPLWS            Yes, get ready
       B    @NEXT            return to GPL

ERROR  BLWP @0               go to Title
                               Screen
       END
```

How do you know where an Assembly Program is loaded in Memory with the Editor/Assembler module?

After you have loaded your program load the Debugger or EXPLORER. Then look at the following memory locations:

>2024 (>A000) First Free Address in Hi-Mem
>2026 (>FFD7) Last Free Address in Hi-Mem
>2028 (>2676) First Free Address in Low-Mem
>202A (>3F38) Last Free Address in Low-Mem

A normal relocatable assembly program with the E/A loads first at >A000. The first load address in Low Mem is >2676.

In Extended Basic the pointers are:

>2002 (24FA) First Free Address in Low-Mem
>2004 (4000) Last Free Address in Low-Mem

So in Extended Basic a normal relocatable assembly program starts to load at >24FA. If the program contains any AORG statements then the pointers WILL NOT be updated by the loader with either module. Also the Extended Basic loader will NOT load an assembly program into Hi-Mem (>A000-FFE7) unless it is AORG there.

What does * WARNING * CONTROL CHARACTER REMOVED PRESS ENTER TO CONTINUE mean when a file is loaded into the Editor/Assembler Editor?

This warning will appear whenever you load a file into the E/A Editor that was previously edited by TI Writer and saved with the SAVE FILE command instead of the PRINT FILE command. The SAVE FILE command saves the TI Writer TABs at the end of the file in Control code format along with the file. If you are loading one of these files do not worry about this message. The editor just removed the Tab settings and the file will Assemble properly.

This warning will also appear if you load a DSI/FIX 80 Object code file that was Assembled with the C (Compressed) option. In this case the editor has removed all of the characters that are not in the normal ASCII range. This is MOST of the object code. So when you edit this file you will find a lot of blank spaces where the object code used to be and this file should NOT be saved since most of it is missing.

Why doesn't Advanced Diagnostics currently work with the Myarc Disk Controller or Myarc's MPES-50 Mini Peripheral Expansion System?

Advanced Diagnostics is its own Disk Controller DSR. In essence you could remove the ROM chips from your disk controller card once Diags is loaded. Currently Diags contains the info for 2 different DSRs of the TI and Corcomp hardware differences. We are looking into the necessary changes to add a third DSR to it ot make it work with the Myarc CARD hardware. The Myarc MPES-50 System is also different in a lot of low level ways, so that requires modifications for a fourth DSR. Whew, its already 24K.

Why can't we Edit the information that is in between the sectors?

TOO DANGEROUS! Also, what is read is not what is written, so you would have to reconstruct the entire track each time. Another problem arises in that most of the Corcomp cards lose DATA on a double density Read Track (their new design is a little better). By too dangerous we mean that all it takes is one BIT out of place and you will NEVER see that track again!!!

How come you didn't place a screen dump feature in EXPLORER?

It was originally specked with a screen dump but as we got into it we discovered that it destroyed TOO MUCH of the environment and as such it required that the buffer be expanded beyond our acceptable limit. EXPLORER was not supposed to be any larger than 16K, to allow more room for your programs. As it ended up it is a VERY COMPACT 18K and no matter what we did we couldn't get it down to 16K and still keep all of its current features. We also felt that since there are a number of memory dump and disassembly programs out that this did not detract from EXPLORER's original purpose "To allow you to watch and experiment with the 4A's internal operations." and "To be one of the best debugger, tracer, single stepper programs around for ANY computer." We believe we met those goals since I don't know of another utility that lets you watch, manipulate and trace so many different items in the Basics or Assembly Language.

What's new in "As The Electron Turns"?

We would like to thank everyone for their vote of support on the Corcomp legal issue. No, they are NOT out of bankruptcy yet and Yes, we are still involved with the time consuming legal dribble. From what we are currently hearing about cards going bad it doesn't sound to good. I also understand that they are now CHARGING $50.00 to repair their defective units!

You can tell when your Disk Controller is starting to go out when it starts to return NO DISKETTE OR NO DRIVE ERRORs or I/O ERROR 06. With their old design this was an annoyance. However, with their new design when this occurs some of our friends have told us that it has erased some of their files off the diskette! This has also happened to us. Bad News. When you are dealing with a mass storage device it should either work or not work. It shouldn't partially go out and start eating files! Maybe they will get it right someday, however, I doubt that they will ever get the few bugs worked out of the software. Since they REFUSE to honor their agreements we aren't going to be doing any work for them. And, it will be awful hard to find the bugs without the source code that we hold the Copyrights on. Bankruptcy or no Bankruptcy that's no way to conduct business. Speaking of which, most companies that we know of never made it out of Bankruptcy.

Many of you have asked about other publications that are SOLELY devoted to the 99/4A computer. Besides all of the Users' Groups newsletters there are two other publications that we are familiar with.

SUPER 99 MONTHLY - is a newsletter type publication that covers a wide diversify of programming which includes Extended Basic, Multiplan, some Forth and some Assembly Language. It is published monthly by Richard Mitchel:

Bytemaster Computer Services
171 Mustang Street
Sulphur, LA 70663

The subscription rates in U.S. funds are:
12.00/year - U.S.
16.00/year - Forgein First class (surface)
26.50/year - Forgein Air Mail

MICROPENDIUM - is a monthly publication that contains Hardware and Software reviews, articles and a lot of advertising for the 99/4A. It is published by John Koloen and Laura Burns:

MICROpendium
P.O. Box 1343
Round Rock, TX 78680

The subscription rates in U.S. funds are:
15.00/year - U.S. Third Class
18.50/year - U.S. First Class
18.50/year - Canadian
21.50/year - Other countries - Surface
28.50/year - Other countries - Air Mail
(Texas residents add 5.125% sales tax)

When will we receive the other issues of the Smart Programmer newsletter?

We are currently looking at producing a large (64 page) newsletter to be mailed out as the next issue(s), hopefully before Christmas. This 64 pager would be the Oct-Jan part of your subscription. It will contain many of the programs and tips and tricks that a lot of you have sent in. This will allow us to complete the 12 issues and get caught up.

Many of you have asked which Commands or Statements are faster than others so we asked Mike McCue to do some timing tests for us. Mike worked on Night Mission with us. Here are his results using three numeric variables A, B & I, two string variables A$, B$ and with three sprites in auto-motion. Each Command or Statement was timed in a FOR I=1 to 1000 Loop. The times listed are in seconds using Extended Basic Version 110.

ABS	- 6	COS	- 78
ASC	- 8	DELSPRITE #	- 15
ATN	- 50	DELSPRITE ALL	- 30
CALL sub	- 7	DISPLAY AT	- 59
CHAR	- 86	DISTANCE # #	- 33
CHARPAT	- 62	DISTANCE # XY	- 35
CHARSET	- 204	EXP	- 87
CHR$	- 16	GCHAR	- 21
CLEAR	- 20	GOSUB	- 1
COINC # #	- 34	GOTO	- 1
COINC # XY	- 36	HCHAR	- 21
COINC ALL	- 15	+ 6 per RPT #	
COLOR #	- 17	IF THEN ELSE	- 4
COLOR	- 18	INT	- 6
JOYST	- 27	!	- 4
KEY	- 24	RETURN	- 1
LEN($)	- 8	RETURN NEXT	- 1
LOCATE	- 22	RETURN #	- 1
LOAD(1,2)	- 17	RND	- 75
LOAD(A,B)	- 23	RPT$	- 30
LOG	- 117	SCREEN	- 10
MAGNIFY	- 11	SEG$	- 27
MAX	- 12	SGN	- 5
MIN	- 12	SIN	- 70
MOTION	- 25	SOUND	- 47
PATTERN	- 17	+ 15 per voice	
PEEK	- 19	SPRITE	- 43
PI	- 5	SQR	- 81
POS	- 22	STR$	- 19
POSITION	- 30	TAN	- 172
PRINT	- 100	VAL	- 21
RANDOMIZE	- 12	VCHAR	- 25
REM	- 4	+ 2 per RPT #	

Why can we have only 4 sprites visible on a row at one time?

This is a hardware property of the TMS 9918A VDP Processor. In this processor there are only 4 Sprite registers so it can not track the fifth sprite on the row. The registers are used to hold position etc. for the current pixel row being drawn.

Is there anyway to use CALL LOAD without having Memory Expansion?

Up until now the answer was NO. The first thing CALL LOAD does is to check to see if CALL INIT has been executed, and if it has not it returns with a syntax error. When CALL INIT is executed it places the value >AA55 at >2006 in Low Memory Expansion. When the CHECK INIT routine is called it checks for >AA55 at >2006 and if it's not there it returns an error. The way around this is to modify Extended Basic with the GRAM KRACKER^{tm} to bypass this test and just execute the CALL LOAD. This can be accomplished by changing the GPL Opcode at Grom/Gram >C044 in Extended Basic from: 06 C1 EB which is CALL CHECK INIT to: 05 C0 47 which is BRANCH TO >C047, the address of the next instruction. The CALL LOAD routine does not need Memory Expansion when you are poking values into Scratch Pad Ram so I don't know why TI locked it out if it isn't there. Maybe they just didn't want us to get in trouble with trying to load an Assembly Program into a non-Memory Expansion system.

Some of you have asked about a German made Extended Basic module that contains many new CALLs and Hi-Resolution Graphics.

This Module is made under licenses by Texas Instruments and Apesoft and is distributed by MECHATRONIC GmbH. It is fully compatible with the current Extended Basic but it has a lot of new enhancements. Many of the New CALLS were added to it by Heiner Martin. They include: GSAVE, GLOAD, BHCOPY, VPEEK, GPEEK, VPOKE, ALLSET, WAIT, MOVE, MLOAD, MSAVE, BYE, NEW, RESTORE, QUITOF, QUITON, SPRON, SPROF, FIND and APESOFT.

CALL APESOFT Loads the Hi-Resolution graphic LINKs into Low Memory Expansion. They allow you to draw and display line and graphs on a pixel by pixel basis. It does not use bit map mode but instead it uses a technique similar to TI Logo with character redefinition. The Demo program we received with the module is very impressive and very small in size compared to what it does! It also is very easy to use, no complicated CHR$(xx) type commands.

Heiner has also written a CONVERT Program that is available in a module. It converts DIS/VAR 80 into RUN-able Extended Basic programs. This allows you to use the TI-Writer or E/A Editor to write your XB programs and then convert them to run in XB.

There are also many new Hardware and Software projects currently in the works over in Germany of the TI 99/4A. I believe that you can obtain information on the above items and the new hardware from the following companies.

For More info on the Extended Basic contact which is approx 120 U.S. Dollars

MECHATRONIC GmbH
Dresdener StraBe 21
7032 Sindelfingen
West Germany
(0 70 31) 87 50 42

For info on the CONVERT Program which is approx 29 U.S. dollars contact:

Elektronik-Service
Linning 37
D-4044 Kaarst 2
West Germany.

Starting with this issue we have Mariusz Stanczak writing the Fourth column. Mariusz is very good with many different versions of Forth and has written a number of programs in TI and Wycove Forth. He has an excellent understanding of the Forth interpreter and the TI Forth Source Code.

In the next issue he has promised to include his unBASVE routine, which will convert a BSAVED application back into normal Forth. We have been giving him your questions on Forth so he will be answering them also. We are very pleased that he is writing the Forth column, which is a big help with the newsletter.

At the end of Mariusz's column we have also included a revision to the TI Forth 40 column editor. This revision was given to us from Pete Korner of the LA 99ers Users Group. It adds Auto-Repeat to the cursor and it defines the function keys so they a closer to the Editor/Assembler editor's function key assignments. On screen 41 you will find two constants for the cursor speed and delay. You can set these to your liking. If you have BSAVED your system you will need to start fresh, load this editor and then all of the options you want and then reBSAVE your system.

This is a nice and much need addition to the TI Forth editor. Thank You Pete.

At the end of Pete's Forth Editor you will find a DSRLNK Routine for the TI Extended Basic environment.

There are so many of these floating around and we have received a number of these from different people that we are not sure who to credit this to, but Thank You one and All. This DSRLNK is very similar to the Editor Assembler DSRLNK that resides in Low Memory Expansion. To use this in your Extended Basic Assembly programs just use the COPY Directive after you have typed it in and saved it as source code. It works the same as the E/A version and it is entered the same way ie:

```
BLWP @DSRLNK
DATA 8
```

for actual DSR Links (DSK, PIO etc) or

```
BLWP @DSRLNK
DATA >A
```

for subprograms like the one listed on page 4 of this issue.

5TH 1- =FORTH

By Mariusz Stanczak

Since your letters are leading in such a direction from time to time, in this edition of 5th 1- we will 'go back to basics', but with some restrictions as I do not think I can be of much help to people who still can not get their copy of the system loaded. The requirements and procedures are very straightforward so I can not imagine anybody failing to follow them. I would complain to the vendor of such an uncooperative system diskette..

Assuming everybody is getting instant gratification in the form of FORTH's friendly 'ok:' most of the time, lets begin, by concentrating on the times of '?'.

First, a few words of information on the most frequently asked questions. There are two variations of FORTH for the 99/4A that I am familiar with. Both follow what is known as the FIG model, which specifies the rules of the logical and partially physical structure of the FORTH system, but there are distinct differences in the way each version implements those rules. The two are; 1) WYCOVE-FORTH which is copyrighted, non-public-domain, and sells for about $50.00. It has has undergone two major revisions, and 2) the TI-FORTH. The latter is what most of you have as it was given away by TI to the Users' Groups who in turn distributed it for a reasonable fee in the $20.00 range. TI-FORTH has had a few changes done to it and I am told of five versions floating around but can not say much about the seriousness of the corrective work done to each version. The version of a given copy was marked, as I am told, on the original diskette given to the User's Groups for redistribution, so try asking them.

TI-FORTH requires memory expansion, disk subsystem and Editor/Assembler cartridge. -NOTE- There is a public domain extended basic loader available -END OF NOTE- WYCOVE-FORTH permits more flexibility as its requirements are; memory expansion and either Extended Basic, Editor/Assembler or Mini Memory cartridge with either cassette or disk subsystem. For those of you who are new to FORTH, TI-FORTH may present a less intimidating environment at first. I have been told that quite a few times but I still say, it is a initial impression. TI-FORTH starts-up with presenting a little user interface that facilitates loading of elective words and a few other esthetic touches. Beyond those, both systems are, one might say, a bare bone implementations with WYCOVE's supplying better (to my taste) developed routines to support the 99/4A environment.

Many of TI-FORTH users write about problems with getting their printers to work, and in most cases the offender seems to be the Axiom Parallax interface working in the parallel mode. Modify scr# 72 line 4 (all screen and line numbers given in reference to the original system diskette) to read:

```
SET-PAB OUTPT F-D" PIO.CR"  OPN      3
```

The solution to another common problem requires a little introduction into the internals of FORTH which, by the way, receive excellent treatment in the FORTH classic, "Starting FORTH" by Leo Brodie. Highly recommended basic reading.

-DIGRESSION-

There are a number of books and publications on FORTH that are worthy of your time and money. They present varied level of difficulty but keep on reading. Things will fall into their proper places in time. The books that I enjoyed most are:

"Thinking FORTH", Leo Brodie, Spectrum Books <a collection of essays on FORTH philosophy and conventions>

"FORTH Programming", Leo J. Scanlon, Blacsburg Group <an introductory text to the 79-STANDARD with many, many useful examples. Differences between the 79-STANDARD and FIG models clearly pointed out>

"FORTH Encyclopedia", Mitch Derick and Linda Baker, Mountain View Press <an essential reference into FIG-FORTH. For advanced programmers and, definitively for all curious>

Then, there are publications of which the foremost is FIG's own **"FORTH DIMENSIONS"** <mostly for advanced programmers>

"Dr. DOBB's Journal" yearly FORTH issue #9, September <many full length articles and complete programs>

"Computer Language" <a monthly publication with a self explanatory title. Periodically publishes discussions and articles on FORTH and FORTH tools as well as many information of general interest. For advanced programmers>

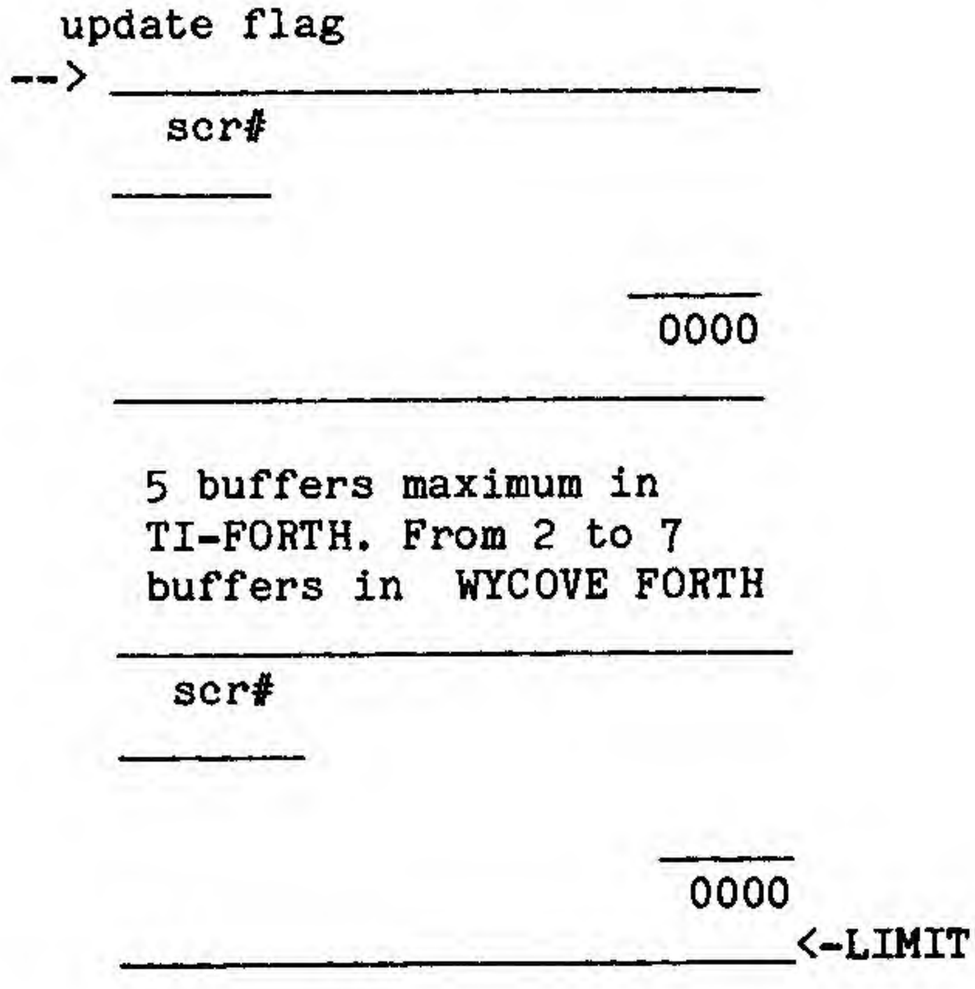
And, at last but not least, the most basic and the most valuable source of information about the particular FORTH system one is using must be the source code. It is invaluable help for people who know assembly language and who think of enhancing their systems, A big help for those who try to gain better understanding of how FORTH works and of what really happens when things do not work as expected. In this area, the biggest surprise awaits those inclined to look, need or just simply desire the access to such information, which for TI-FORTH is available for about $15.00. Wycove Systems Limited wants you to dish out a cool $100.00 for the source code of the interface between the public domain FIG version (developed for the TMS9900 processor by 9900 Software Services and available through FIG for $15.00) and the 99/4A. A quite distasteful proposition if you consider that 80% of that $100.00 source code was copied from the FIG's release. I would think that, more effort was consumed at producing the wealth of tools supplied with the initial system diskette of this otherwise attractive product (it placed fifth out of fifteen in a recent test of FORTH implementations on different machines conducted by a british computer magazine), than was spent writing the 1.5K or so of assembly code that went into the development of the disk and screen I/O routines making the asking price completely unjustifiable.
-END OF DIGRESSION-

This next common problem is with copying screens between floppies which arises when the screen numbers need to be changed also i.e. the screens are moved to different locations. Owners of multi drive systems running TI-FORTH do not have a problem at all. TI has supplied words SCOPY and SMOVE. Consult the system manual; chapter 5, page 7.

-NOTE- The least confusing way to do it is to keep the value in OFFSET equal to 0 (by typing DR0 which sets drive 0 as the system drive) and by referring to screens by their absolute numbers in reference to screen 0 in drive 0, i.e. screens 0-89 are in drive 0, 90-179 in drive 1 and screens 180-269 are in drive 2. This amounts to adding the proper offset yourself, which in TI-FORTH equals 90 for drive 1 and 180 for drive 2. Also, do not forget to set the user variable DISK_HI to the appropriate value for the number of drives on your system, which is 180 for two drives, and 270 for three. -END OF NOTE-

For one drive systems the easiest would be to modify the word SMOVE to prompt you when to change diskettes, or see the word SSMOVE below.

FORTH employs a virtual memory technique in it's disk access. Lets take a closer look at how it happens that we find the screens available to us for editing, or for whatever we want to do with them. The area into which the screens are loaded is called buffer area, which depending on the system may contain any number of buffers (minimum 2 in the FIG model). Each buffer consists of a one word header, a 1024 byte data portion and a terminator word. The portion of buffer essential to our solution is the header which contains the block number and an update flag in the high order bit (the sign bit). The buffers are located in memory as a contiguous array but are treated logically as a circular array. The word +BUF has this function of returning the address of next buffer in this circular array and it is called by BLOCK, PREV, UPDATE AND BUFFER. The buffer area looks something like this:



When we type scr# BLOCK , the word BLOCK will add the value stored in the user variable OFFSET to the value on the stack (scr#), check the buffer area for the presence of the arrived at screen number. If the screen is found then the address of its data area is left on the stack and the execution of BLOCK ends. If the screen is not found in the memory, BLOCK will call BUFFER which will check the update flag of the buffer pointed to by another user variable USE. If the flag is set, BLOCK will request (word R/W) to write such marked block to disk prior to calling the screen we requested from disk which is written into such freed buffer, again pushing the address of its data area on the stack. All we have to do, to accomplish moving of screens around, is to change the screen number in the control word which is located at the value left on the stack by BLOCK minus two bytes, UPDATE the buffer and the buffer management mechanism will take care of the rest for us, or its action can be forced by word SAVE-BUFFERS.

WYCOVE-FORTH users have the equivalent of SCOPY supplied on screen 58. The word is called COPY (which in slightly modified form you have it in the listing below) and while using it, do not forget about the proper offset which is equal to 0 for disk #1, 2000 for disk #2, 4000 for disk #3, 6000 and 8000 for cassette drive #1 and #2 respectively. The word SMOVE has to be written, and it could look like this:

```
: SCOPY  ( from+offset to+offset --- )
    SWAP  BLOCK 2-  !  UPDATE ;
: DOCOPY+  ( from to hi_limit lo_limit )
    DO  OVER OVER  SCOPY
    1+  SWAP 1+  SWAP  LOOP ;
: SMOVE  ( from+offset to+offset cnt --- )
   DUP  0>
   IF  R/W-CLOSE  >R  OVER OVER  <
     IF                      \ move back>front
       R 1- + SWAP R 1- + SWAP
       R> 0
       DO  OVER OVER  SCOPY
         1-  SWAP 1-  SWAP
       LOOP
     ELSE                    \ move front>back
     R> 0  DOCOPY+
     THEN
   ELSE DROP
   THEN  DROP DROP  SAVE-BUFFERS ;
```

For one drive systems, the above gets a little bit more complicated. I have written it as a multiple DO-LOOP structure to keep track of when to prompt for changing the floppies, but try rewriting it as a recursive construct using MYSELF (called RECURSE in earlier WYCOVE's implementation). The following will work with either TI's or WYCOVE's system installed.

```
: ?BUFFERS  ( --- number_of_buffers )
    1  PREV @  BEGIN  +BUF
               WHILE  SWAP  1+  SWAP
               REPEAT
    DROP ;
: MSG1  ( --- )
    ." Insert source disk"
    KEY DROP  CR ;
: MSG2  ( --- )
    ." Insert destination disk"
    KEY DROP  CR  SAVE-BUFFERS ;
: SSMOVE  ( from to cnt --- )
    DUP  0>
    IF  >R >R >R  ?BUFFERS
      R> R> R>  4 PICK  /MOD  -DUP
      IF  CR  SWAP  >R 0
        DO  R/W-CLOSE  MSG1  3 PICK  0
          DOCOPY+  R/W-CLOSE  MSG2
        LOOP  R>
      THEN  -DUP
      IF
        R/WCLOSE  MSG1  0
        DOCOPY+  R/W-CLOSE  MSG2
      THEN
    THEN  DROP DROP DROP ;
```

To load the above definitions your system will have to have the following defined:

in **TI-FORTH--**

```
: 0>  ( n --- f )  0  > ;
CODE PICK  ( item# --- item ) HEX
 C019 , 0A10 , A009 , C650 , 045F , DECIMAL
```

in **WYCOVE-FORTH--**

```
CREATE PICK  ( item# --- item )  SMUDGE HEX
 C016 , 0A10 , A006 , C590 , 0459 , DECIMAL
```

-NOTE- This version of the word PICK, which is required in the 79-STANDARD, does not do the parameter stack range check. It counts the items on stack starting with one i.e 1 PICK (not 0 PICK) is equal to DUP. -END OF NOTE- ...and end of this edition of 5th 1-. Until the next time around. Happy FORTHing. MS

```
SCR #34
  0 ( SCREEN EDITOR 09JUL82 LCT) 0 CLOAD RKEY
  1 BASE->R DECIMAL 33 R->BASE CLOAD RANDOMIZE
  2 BASE->R   HEX VOCABULARY EDITOR1 IMMEDIATE EDITOR1 DEFINITIONS
  3   : BOX 8F7 8F1 DO 84 I VSBW LOOP ;
  4   : CUR R# ;
  5   : !CUR 0 MAX B/SCR B/BUF * 1- MIN CUR ! ;
  6   : +CUR CUR @ + !CUR ;
  7   : +LIN CUR @ C/L / + C/L * !CUR ;
  8   0 VARIABLE S_H     DECIMAL
  9   : FTYPE 40 * 124 + SWAP VMBW ;
 10   : LISTA 0 0 GOTOXY DUP SCR ! ."  SCR# " DUP .
 11   ." (decimal  " DECIMAL . ." )"   CR CR CR
 12    16 0 DO I 3 .R CR LOOP ;     : ROWCAL S_H @ IF 29 + ENDIF ;
 13   : LINE. DO I SCR @ (LINE) DROP ROWCAL 35 I FTYPE LOOP ;
 14   : LISTB L/SCR 0 LINE. ;
 15 R->BASE -->

SCR #35
  0 ( SCREEN EDITOR 09JUL82 LCT)
  1 : LISTL BASE->R LISTA 4 1  GOTOXY
  2 ."          1         2         3     " 4 2 GOTOXY
  3 ." ....+....0....+....0....+....0....+" 
  4 0 S_H ! LISTB R->BASE ;
  5 : LISTR BASE->R DROP 4 1 GOTOXY
  6 ." 3         4         5         6     " 4 2 GOTOXY
  7 ." 0....+....0....+....0....+....0...." 
  8 1 S_H ! LISTB R->BASE ;
  9 : BCK  0 L/SCR 2+ GOTOXY QUIT ;
 10 : PTR SCR @ B/SCR * CUR @ B/BUF /MOD ROT + BLOCK + ;
 11 : R/C CUR @ C/L /MOD ;  ( --- COL ROW )
 12 : DELHALF PAD 64 BLANKS PTR PAD C/L R/C DROP - CMOVE ;
 13
 14 -->
 15


SCR #36
  0 ( SCREEN EDITOR 12JUL82 LCT) BASE->R  DECIMAL
  1    : .CUR CUR @ C/L /MOD 3 + SWAP 4 + DUP S_H @
  2      IF 32 > IF 29 - ELSE SCR @ LISTL ENDIF
  3      ELSE 39 < 0= IF SCR @ LISTR 29 - ENDIF
  4      ENDIF SWAP GOTOXY ;
  5    : DELLIN R/C SWAP MINUS +CUR PTR PAD C/L CMOVE DUP L/SCR SWAP
  6      DO PTR 1 +LIN PTR SWAP C/L CMOVE LOOP
  7      0 +LIN PTR C/L 32 FILL C/L * !CUR ;
  8    : INSLIN R/C SWAP MINUS +CUR L/SCR +LIN DUP 1+ L/SCR 0 +LIN
  9      DO PTR -1 +LIN PTR SWAP C/L CMOVE -1 +LOOP
 10      PAD PTR C/L CMOVE C/L * !CUR ;
 11    : RELINE R/C SWAP DROP DUP 13 EMIT LINE. UPDATE .CUR ;
 12 : +.CUR +CUR .CUR ;
 13 : TAB PTR DUP @ 32 = 0= IF BEGIN 1+ DUP 1 +CUR C@ 32 = UNTIL
 14   ENDIF CUR @ 1023 = IF .CUR 1 ELSE  BEGIN 1+ DUP 1 +CUR C@ 32 >
 15   UNTIL .CUR ENDIF ; R->BASE -->
```

```
SCR #37
  0 ( SCREEN EDITOR 12JUL82 LCT)  BASE->R  DECIMAL
  1 : -TAB PTR DUP C@ 32 > IF BEGIN 1- DUP -1 +CUR C@ 32 = UNTIL
  2   ENDIF BEGIN CUR @ IF 1- DUP -1 +CUR C@ 32 > ELSE .CUR 1 ENDIF
  3   UNTIL BEGIN CUR @ IF 1- DUP -1 +CUR C@ 32 = DUP IF 1 +.CUR
  4   ENDIF ELSE .CUR 1 ENDIF UNTIL ; : !BLK PTR C! UPDATE 1 +.CUR ;
  5  : BLNKS PTR R/C DROP C/L SWAP - 32 FILL ;
  6 : FLIP S_H @ IF -29 ELSE 29 ENDIF +.CUR ;
  7 : REDRAW SCR @ S_H @ IF LISTR ELSE LISTL ENDIF UPDATE .CUR ;
  8 : NEWSCR 0 SWAP LISTL !CUR .CUR ;
  9 : +SCR SCR @ 1+ NEWSCR ;
 10 : -SCR SCR @ 1- 0 MAX NEWSCR ;
 11 : DEL PTR DUP 1+ SWAP R/C DROP C/L SWAP - CMOVE 32
 12   PTR R/C DROP - C/L + 1- C! ;
 13 : INS 32 PTR DUP R/C DROP C/L SWAP - + SWAP DO
 14   I C@ LOOP DROP PTR DUP R/C DROP C/L SWAP - + 1- SWAP 1- SWAP
 15   DO I C! -1 +LOOP ; R->BASE -->
```

```
SCR #38
  0 ( SCREEN EDITOR 12JUL82 LCT) BASE->R HEX  29 LOAD
  1 : VED BOX SWAP CLS LISTL !CUR .CUR BEGIN  RKEY  CASE
  2   0F OF BCK                ENDOF  01 OF DELHALF BLNKS RELINE ENDOF
  3   08 OF -1 +.CUR           ENDOF  0C OF +SCR                 ENDOF
  4   0A OF C/L +.CUR          ENDOF  02 OF -SCR                 ENDOF
  5   0B OF C/L MINUS +.CUR ENDOF  03 OF DEL RELINE             ENDOF
  6   09 OF 1 +.CUR            ENDOF  04 OF INS RELINE           ENDOF
  7   0D OF 1 +LIN .CUR        ENDOF  07 OF DELLIN REDRAW        ENDOF
  8   0E OF FLIP               ENDOF  06 OF INSLIN REDRAW        ENDOF
  9   1E OF INSLIN BLNKS REDRAW ENDOF 16 OF TAB                ENDOF
 10   7F OF -TAB ENDOF
 11   DUP 1F > OVER 7F < AND IF DUP EMIT DUP !BLK ELSE   7 EMIT
 12   THEN   ENDCASE   AGAIN   ;  FORTH DEFINITIONS
 13 : WHERE EDITOR1 B/SCR /MOD SWAP B/BUF * ROT + 2- VED ;
 14 : EDIT EDITOR1 0 VED ;  : ED@ EDITOR1 SCR @ EDIT ;
 15 R->BASE
```

```
SCR #41
  0 ( EDITOR REPEAT KEY ROUTINE Pete Korner 12/3/84)
  1   BASE->R DECIMAL   0 VARIABLE MY
  2 : BLINK CURPOS @ DUP VSBR MY C!
  3   3 0 DO DUP 30 SWAP VSBW LOOP MY C@ SWAP VSBW ;
  4   4 CONSTANT W ( repeat speed) 30 CONSTANT X  ( delay )
  5   0 VARIABLE Y   X VARIABLE Z  0 VARIABLE OK
  6 : RKEY  BEGIN ?KEY -DUP   BLINK BLINK
  7   IF Y @   1  Y +! IF Z @  Y @ <
  8   IF W   Z !   1   Y !
  9   1 ELSE  OK @  OVER = IF  DROP 0
 10   ELSE 1 DUP   Y !   THEN THEN
 11   ELSE  1   THEN
 12   ELSE  X   Z !  0   Y !
 13   0 THEN  UNTIL DUP OK ! ;  R->BASE
 14
 15
```

Here is the DSRLNK Routine for use in the Extended Basic environment that we promised you last issue. I'm not sure where this one came from, there are so many of them floating around. It is basically the same one used in the Editor Assembler Module.

```
*--------------------------------------------------------------*
* DSRLNK routine for XB - Save as source code and add it to your desired  *
* Assembly program with the COPY directive.                    *
* VSBR equate must be included in the Calling Program or DSRLNK routine . *
*--------------------------------------------------------------*
VSBR    EQU  >2028              VDP single byte read XB= >2028
PNTR    EQU  >8356              Pointer to search name address
SCLEN   EQU  >8355                          in VDP XB= >8356
CRULST  EQU  >83D0
SADDR   EQU  >83D2
GPLWS   EQU  >83E0              GPL/Extended Basic workspace
SAVCRU  DATA 0                  CRU address of peripheral
SAVENT  DATA 0                  Entry address of DSR
SAVLEN  DATA 0                  Save device name length
SAVPAB  DATA 0                  pointer to device name in PAB
SAVVER  DATA 0                  Version number of DSR
DLNKWS  DATA 0,0,0,0,0
TYPE    DATA 0,0,0,0,0,0,0,0,0,0,0
NAMBUF  DATA 0,0,0,0
H20     DATA >2000
DECMAL  TEXT '.'
HAA     BYTE >AA
*--------------------------------------------------------------*
* DSRLNK - Workspace,Program Counter for BLWP @DSRLNK *
*--------------------------------------------------------------*
DSRLNK  DATA DLNKWS,DLENTR
DLENTR  MOV  *R14+,R5           Fetch program type for link
        SZCB @H20,R15           Reset equal bit
        MOV  @PNTR,R0           Fetch pointer into PAB
        MOV  R0,R9              Save pointer
        AI   R9,-8              Adjust pointer to flag byte
        BLWP @VSBR              Read device name length
        MOVB R1,R3              Store it
        SRL  R3,8               Make it a word value
        SETO R4                 Initialize counter
        LI   R2,NAMBUF          Point to NAMBUF
LNK$LP  INC  R0                 Point to next char of name
        INC  R4                 Increment char counter
        C    R4,R3              End of name ?
        JEQ  LNK$LN             Yes
        BLWP @VSBR              Read current char
        MOVB R1,*R2+            Move it to NAMBUF
        CB   R1,@DECMAL         Is it a decimal point ?
        JNE  LNK$LP             No
LNK$LN  MOV  R4,R4              Is name length zero
        JEQ  LNKERR             Yes - Error
        CI   R4,7               Is name length > 7 ?
        JGT  LNKERR             Yes - Error
        CLR  @CRULST
        MOV  R4,@SCLEN-1        Store name length for search
        MOV  R4,@SAVLEN         Save device name length
        INC  R4                 Adjust it
        A    R4,@PNTR           Point to position after name
        MOV  @PNTR,@SAVPAB      Save pointer into device name
```

```
****** SEARCH ROM FOR DSR
SROM    LWPI GPLWS              Use GPL workspace to search
        CLR  R1                 Version found of DSR
        LI   R12,>0F00          Start over again
NOROM   MOV  R12,R12            Anything to turn off ?
        JEQ  NOOFF              No
        SBZ  0                  Yes, turn it off
NOOFF   AI   R12,>0100          Next DSR ROM's turn on
        CLR  @CRULST            Clear in case we are finished
        CI   R12,>2000          At the end ?
        JEQ  NODSR              No more ROM's to turn on
        MOV  R12,@CRULST        Save address of next CRU
        SBO  0                  Turn on ROM
        LI   R2,>4000           Start at the beginning
        CB   *R2,@HAA           Is it a valid ROM ?
        JNE  NOROM              No
        A    @TYPE,R2           Go to the first pointer
        JMP  SG02
SG0     MOV  @SADDR,R2          Continue where we left off
        SBO  0                  Turn ROM back on
SG02    MOV  *R2,R2             Is address a zero ?
        JEQ  NOROM              Yes, no program to look at
        MOV  R2,@SADDR          Remember where to go next
        INCT R2                 Go to entry point
        MOV  *R2+,R9            Get entry address

****** CHECK AND SEE IF NAME MATCHES
        MOVB @SCLEN,R5          Get length as counter
        JEQ  NAME2              Zero length ? Don't do match
        CB   R5,*R2+            Does length match ?
        JNE  SG0                No
        SRL  R5,8               Move to right place
        LI   R6,NAMBUF          Point to NAMBUF
NAME1   CB   *R6+,*R2+          Is character correct ?
        JNE  SG0                No
        DEC  R5                 More to look at ?
        JNE  NAME1              Yes
NAME2   INC  R1                 Next version found
        MOV  R1,@SAVVER         Save version number   ¦ Could be used to
        MOV  R9,@SAVENT         Save entry address    { avoid another lookup
        MOV  R12,@SAVCRU        Save CRU address      ¦ on subsequent calls
        BL   *R9                EXECUTE ROUTINE
        JMP  SG0                Not right version
        SBZ  0                  Turn off ROM
        LWPI DLNKWS             Select DSRLNK workspace
        MOV  R9,R0              Point to flag byte in PAB
        BLWP @VSBR              Read flag byte
        SRL  R1,13              Just want the error flags
        JNE  IOERR              Error !
        RTWP

****** ERROR HANDLING
NODSR   LWPI DLNKWS             Select DSRLNK workspace
LNKERR  CLR  R1                 Clear the error flags
IOERR   SWPB R1
        MOVB R1,*R13            Store error flags in calling R0
        SOCB @H20,R15           Indicate an error occurred
        RTWP                    Return to caller
*       END                     Only if not assembled by COPY directive.
```

PC NOTES

A number of you have asked about the software we use on the TI PC. There are two main pieces of software used for our business.

For Word Processing we are currently using Easy Writer II with Easy Speller II. It is not the most powerful word processor around but, it is very easy to use. It is a File/ Document/ Page type word processor in that you name a file folder which can hold approximately 100 documents of any number of pages. Once a file folder is opened you select the document and the page to edit. If you do not select a page number to edit it defaults with page 1. For example: This page is in file folder F:SPNL84, document number 9 - September 84, page 25.

Easy Writer II has served us quite well and we have used it to produce all of our newsletters, brochures, manuals and letters.

For a Data Base we are currently using dBASE II but we are looking to upgrade to dBASE III. Do we highly recommend dBASE? Well, lets just say that for us it gets the job done. There are a lot more Data Base type programs out there that are MUCH easier to set up and use. dBASE, in our opinion, requires an understanding of Pascal programming to take FULL advantage of its power. And even with that, dBASE II still has some problems. First off its SLOW on its screen I/O and VERY SLOW when it packs and re-indexes an entire large data base. Luckily, packing and re-indexing is done only a couple of times a year. When dBASE packs a data base it removes all records marked for deletion. Indexing is performed to build an index file for quick FINDs. When we Pack and Index on Name and Index on Zip code it takes approximately 4 hours to complete. We've been told that dBASE III is much faster on its Sorts, Packs and Indexing.

The next problem is the fact that dBASE II executes the command files, that you generate in a Pascal like fashion, interpretively. Unfortunately it does not tokenize the command files so it must interpret ASCII files each time they are run, this gives it the blinding speed of TI BASIC. No, maybe TI BASIC is faster. If the speed isn't enough to annoy you then we're sure that the little quirks in the way the interpreter handles variables and strings will raise your blood pressure. The next problem is that I do not care for the heavy structured programming of Pascal. I miss the GOTOs, GOSUBs, JMPs, BLs and BLWPs, of the Basics and Assembly. Without them I find that there is too much duplication of code that could very easily be a subroutine.

By now you are probably asking WHY we are using this software if we dislike it so much. Well it has its good points. It is very flexible, so once you have mastered its command files you can build a complete custom business system with it. Over the past few years we have set it up for our mail order business with the following features. Order Entry & Invoicing, Brochure Requests, Subscription Entry - Tracking - and Label Printing - sorted by Country and Zip for bulk mailing. Automatic Deposit slip entry, printing and reporting which is tied to our book keeping for income reports. Automatic end of the month and end of the year book keeping updates and reports including sales reports for each item. It also has many other editing, label and report printing and book keeping features built into it. So at this point we are very used to it and as such we do not want to start all over again with another system, thus the to upgrade to dBASE III.

Do we recommend it? Only if you like Pascal programming, have lots of time to get it set up, and enjoy programming. Or, if you have LOTS of money to pay a freelance dBASE programmer. For every 100 dBASE programs sold, I'm willing to bet that only 10-20 of them are in actual everyday use. Personally we think that the success of dBASE has more to do with the advertising hype and some strange prestige thing about owning it. It wasn't due to it's user friendliness or its blinding speed. There are a lot of other Data Base programs available for the PC that are easier to set up and use and many of them are faster and will hold more records than dBASE II. So before you buy one, take a GOOD look at all of them.

Our other software includes things like Cross Talk for our Terminal Emulator package, Microsoft Macro Assembler for assembly, Forth, and some other utilities.

SUBSCRIPTION INFORMATION

THE SMART PROGRAMMER - a monthly 16+ page newsletter published by **MILLERS GRAPHICS**
U.S. 12.50 year - Foreign Surface Mail 16.00 year - Foreign Air Mail 26.00 year

Back issues are available. We can start your subscription with the FEB. 84 issue
To subscribe send a Check, Money Order or Cashiers Check, payable in U.S. currency

TO: **MILLERS GRAPHICS**
1475 W. Cypress Ave.
San Dimas, CA 91773

MILLERS GRAPHICS
1475 W. Cypress Ave.
San Dimas, CA 91773

BULK RATE
U.S. POSTAGE
PAID
San Dimas, CA 91773
PERMIT NO. 191